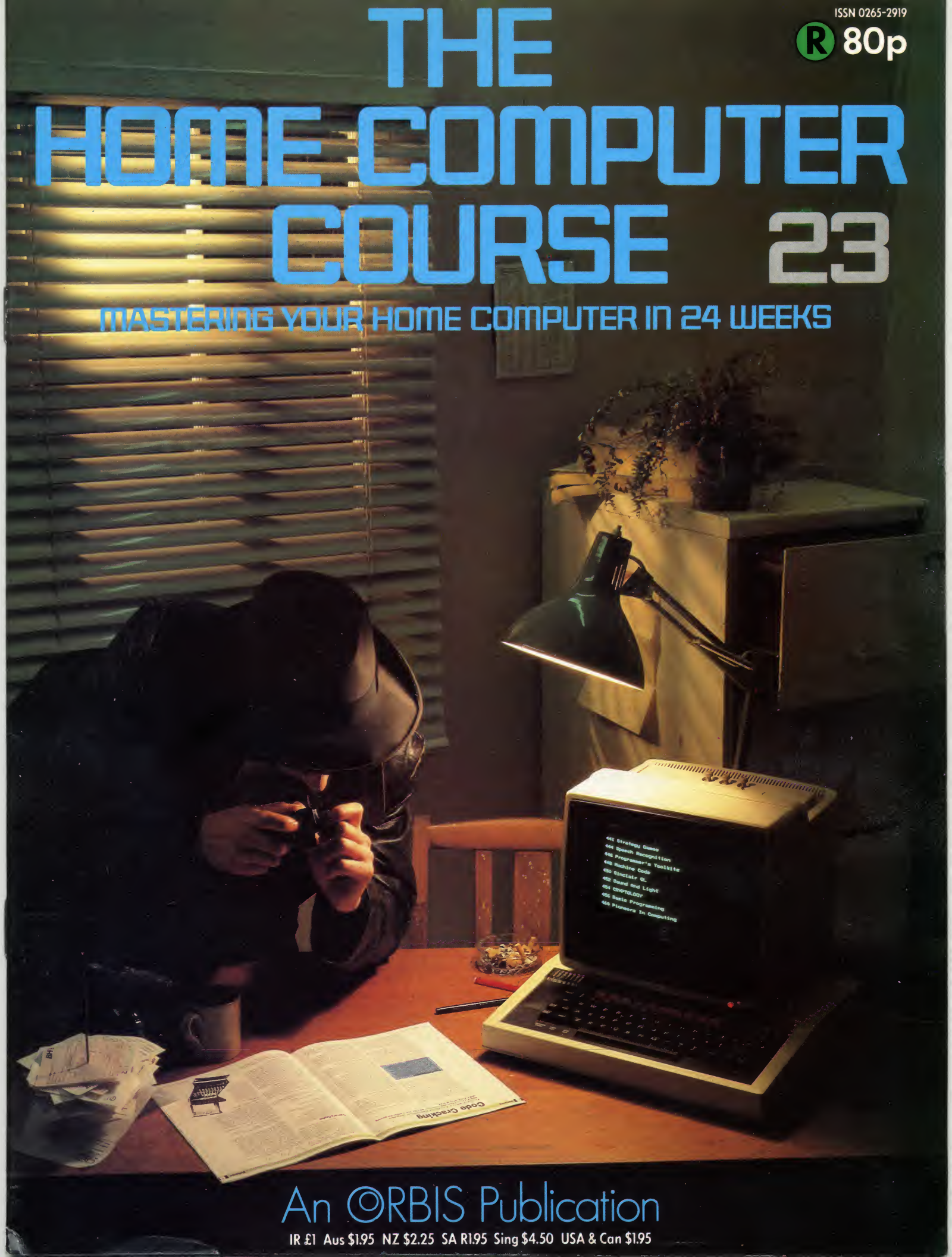
ISSN 0265-2919

80p

# THE HOME COMPUTER COURSE 23

## MASTERING YOUR HOME COMPUTER IN 24 WEEKS




An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

# CONTENTS


## Hardware Focus

**Sinclair QL** The machine that embodies all the latest home computer features — 450

## Software

**Tools Of The Trade** Software add-ons that transform your micro's potential by extending its Basic — 444

**Code Cracking** Deciphering codes is one of the most enduring applications of computers — 454

## Basic Programming

**A Matter Of Style** Now that we've completed the address book program, we look at some of Basic's more sophisticated features — 456

## Insights

**War And Peace** With battle simulations you can test your tactical skills against friend, foe or computer — 441

**Word Of Command** How computers can be programmed to recognise human speech — 446

## Passwords To Computing

**Machine Code** Basic is versatile and easy to understand, but machine code is the most direct way to address your micro's memory — 448

## Pioneers In Computing

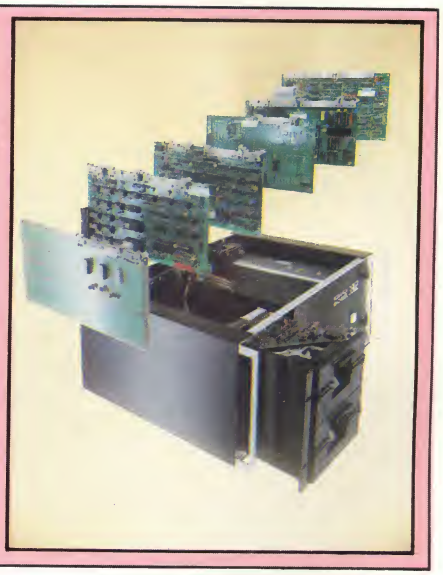
**University Challenge** Manchester dominated the race to build the world's first stored program computer — 460

## Sound And Light

**Sound Principles... Light Refreshments** The Atari has a special chip for producing sound, and the Oric-1 has flexible graphics capabilities — 452


## Next Week

- **The Research Machines 380Z is one of the most popular machines in schools. The reasons for its popularity include high resolution graphics and a wide range of programming languages**
- **We look at the home computer of the future and the facilities it's likely to include. Some will make use of radical new technology, while others will reintroduce features from the earliest machines**



p.450

p.446

**Editor** Richard Pawson; **Consultant Editor** Gareth Jefferson; **Art Director** David Whelan; **Production Editor** Catherine Cardwell; **Staff Writer** Roger Ford; **Picture Editor** Claudia Zeff; **Designer** Hazel Bennington; **Art Assistant** Liz Dixon; **Sub Editors** Robert Pickering, Keith Parish; **Researcher** Melanie Davis; **Contributors** Tim Heath, Henry Budgett, Brian Morris, Lisa Kelly, Steven Colwill, Richard King; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd; Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1; **© 1984 by Orbis Publishing Ltd: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

**HOME COMPUTER COURSE** – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

**How to obtain your copies of HOME COMPUTER COURSE** – Copies are obtainable by placing a regular order at your newsagent.

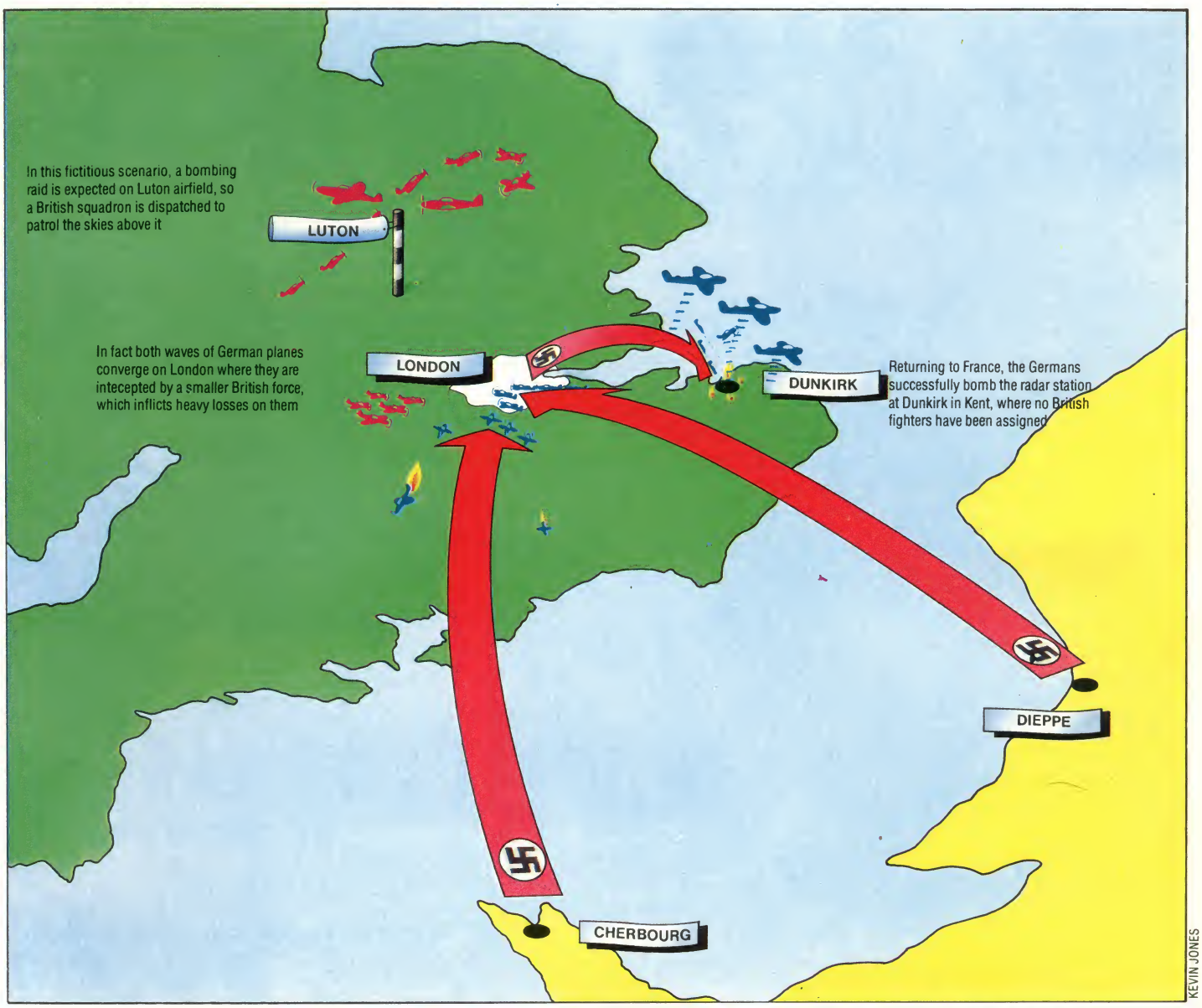
**Back Numbers UK and Eire** – Back numbers are obtainable from your newsagent or from HOME COMPUTER COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

**How to obtain binders for HOME COMPUTER COURSE** – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 4, 5 and 6. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595 Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

**Note** – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

COVER PHOTOGRAPHY BY MARCUS WILSON-SMITH

# War And Peace


In this fictitious scenario, a bombing raid is expected on Luton airfield, so a British squadron is dispatched to patrol the skies above it
LUTON
In fact both waves of German planes converge on London where they are intecepted by a smaller British force, which inflicts heavy losses on them
LONDON
DUNKIRK
Returning to France, the Germans successfully bomb the radar station at Dunkirk in Kent, where no British fighters have been assigned
DIEPPE
CHERBOURG
KEVIN JONES

## You can now purchase games that will test your skills as military strategist and tactician in both historical and fantasy battle simulations

Today modern generals place a great deal of importance on the war games that they play to test planned responses to anticipated attack or 'Threat Scenarios'. To play these sophisticated games complex hardware and software systems have been developed to simulate all the known aspects of a potential conflict, such as the initial deployment of friendly and enemy forces, supply states, availability of reserves, and so on. The system also allows for adverse weather conditions, changes in enemy tactics, the effects of fifth column activities, or any number of variables that could possibly affect the successful execution of a military operation. One of the main functions of

**Battle Of Britain**
Fighter Command (from Strategic Simulations Inc. for the Apple) is typical of the sophisticated strategic war games available for microcomputers. Before the game commences, the player must select from a very wide range of options, including the type of aircraft used and the weather. Play is displayed in the form of moving symbols on the map, with additional information displayed as text. The packaging and documentation include a printed map with cardboard pieces for additional visual reference

IAN McKINNELL

**Tactical Armour Command**
TAC (from Avalon Hill) is available for the Atari, Apple, IBM PC and Commodore 64 computers and simulates armoured conflict during the Second World War. TAC can be played by one or two players who select from five different scenarios, and manipulate British, American, Russian and German forces

the NORAD radar defence system in the Cheyenne Mountains, Wyoming (featured in the film *War Games*) is to continually assess, update, and evaluate the relative capabilities of the United States and the Soviet Union and to aid in the preparation of a response to any new developments.

Of course, war gaming for the amateur general has been somewhat less sophisticated. In order to create the appropriate degree of complexity, the war gamer has had to resort to sheets of tables, voluminous books of rules and innumerable dice. The sheer amount of hard work necessary to play war games has tended to restrict their appeal to a relatively small group of enthusiasts. However, with the arrival of the home computer and the availability of war game programs, all the tedious 'staff work' has disappeared and left in its place an absorbing game that offers both excitement and challenge, the equal of any other type of computer game currently on the market.

An immense variety of games are available. It is possible to recreate or simulate practically any type of warfare from the Ancient Greek to a theoretical clash between NATO and the Warsaw Pact sometime in the future. You can fight air battles, sea battles, wars in outer space, and even wars between mythological empires. The scope is limitless.

Historical games give you the chance to discover where Napoleon went wrong at Waterloo or attempt to outwit Hitler by thwarting his invasion of Russia in 1941. The games set in outer space offer even more opportunity for invention. Not only do you manoeuvre fleets of starships round the galaxy, you also specify the type of ships you want. Compromises have to be made, of course. If you want more speed you may have to sacrifice weapon systems, and better protective screens could reduce the fuel supply. You have to choose the compromise that best suits your style of fighting a campaign.

Unlike their conventional counterparts, computer strategy games need no special skill or knowledge, and most come with short notes and hints for beginners. However, it's worth knowing

COURTESY OF SOFT

IAN McKINNELL

**Legionnaire**
This simulation of warfare between Caesar's forces (you) and the barbarians (the computer — Apple or Atari) is played in real time. Infantry, cavalry and other forces are represented by symbols, which can be selected and moved by means of the joystick-controlled cursor (white square). The game is produced by Avalon Hill

that some games are classified as introductory, intermediate or advanced. If you are thinking of taking up strategy gaming you might be better advised to start at the introductory level, and pick up the basic concepts of war gaming and strategic simulation before advancing to games at the higher levels.

The game format varies to suit the differing needs of the various types of warfare represented. But, in general, the games are played across large maps. Where the map is too large to be displayed on a single screen, then the screen usually acts as a window that can be moved (by means of a joystick) across the map. For an historical simulation the game designers will have tried to reproduce, as faithfully as possible, the terrain over which the original battle was fought. In 'Computer Bismarck' by SSI, the action takes place mainly in the North Atlantic, and this did not pose too many problems in terms of graphics design. On the other hand, for another SSI game, 'Battle for Normandy', the designers' task was far more difficult. Not only did the general terrain have to be correct but so did specific features such as the beaches and the coast, the towns, villages and rivers. For non-historical games the designer has more scope to produce variety for the player to make full use of the forces he has available, but even here the designer must be careful to include sufficient checks and balances to prevent the game becoming too easy for one side or the other.

The map also has a grid superimposed on it. This grid subdivides the map in the same way that a chess board is divided into squares, though the grid on a war games map is often hexagonal rather than square. Each square or hexagon is given a value according to the type of terrain contained within it. This value represents the degree of difficulty a unit would have moving into or through that particular area. The effort of moving through the area would cause the movement allowance of the unit to be reduced by the corresponding value. When the movement allowance of the unit equals zero, or is less than the value of the area it is proposing to enter, it may not move any further that turn.

The game is usually divided into a number of 'game turns' that represent elapsed time, and each player is given a number of objectives that must be accomplished in the time available in order to win the game. In most cases it is not necessary, or even possible, to achieve all the objectives set. So the first decision the player has to make is to assess his chances and determine his strategic priorities accordingly. In such scenarios the role of the opponent is often to stop the attacker from gaining his set objectives. Once again it is probably not possible to protect everything, so the defender must decide when to abandon hopeless positions, how long to cling to strongholds, and whether or not to take the risk of launching counter-attacks to regain lost positions or disrupt his opponent's preparations for a fresh attack.

The player communicates with the program through the graphic and textual representation of the forces under his command that are on the map. The graphic display represents the location of a particular unit on the battlefield and the textual display supplies information relating to the unit's combat efficiency, and movement allowance. The player moves his units by nominating them with a cursor or by having the computer present them to him in rotation. Once a unit has been nominated, the command to move the unit is given. In the case of a map with a hexagonal grid, 1 would send the unit north, 2 would send it north-east, and so on around the points of the compass. An increasing number of these games work with joysticks or trackballs, in which case the unit can simply be 'picked up' and moved in the desired direction. To terminate the movement of an individual unit the command FINISH or F is often used. Even then some games will allow the player to renominate the unit and move it again, unless its movement allowance has been exhausted. When all movement has been completed, the player indicates the fact to the computer with the command EXECUTE or E. The computer will then initiate the combat phase.

COURTESY OF SOFT

**Eastern Front**

In this game the player takes on the role of the German army trying to reach Moscow in 1941, whilst the computer plays the defending Russian forces. Written by Chris Crawford and distributed by Atari, it incorporates radical new features, such as 'fine scrolling'

One of the most visually pleasing features of Eastern Front is the way the map changes as the year progresses. In autumn the forests defoliate, and then in winter the rivers freeze up and the ground is covered in snow. It is considered extremely difficult to reach Moscow in this game

IAN McKINNELL

During the combat phase the computer will indicate the friendly units that are in a position to engage the enemy and provide information about the relative strengths of the units involved. On the basis of this information, the player may accept or reject each combat suggestion as it is offered to him. Once all the combat has been resolved, and the effects calculated and displayed, the second player begins his turn.

For many people, the fascination of strategic games arises from there being no one 'correct' solution to the problems that are posed by the game. The player's enjoyment is derived from overcoming the physical and logistical problems of the terrain that he is operating in as well as meeting the intellectual challenge of using the resources available to defeat the enemy. Naturally, every strategist would like to win by using the most daring schemes and carefully laid traps, but above all the strategist wants to win!

# Tools Of The Trade

**Tool kits are software packages that will enhance a limited dialect of Basic and offer de-bugging facilities to the programmer**

Early home computers, such as the Apple II and Commodore PET, had limited capabilities and were designed primarily to manipulate numbers and text. The BASIC supplied for these machines was required only to provide commands and routines for these purposes. As a result, many 'utility' or 'tool kit' programs were written, usually in machine code, that operated from outside the BASIC programming area. These provided programming aids in the form of additional direct commands that could help in program construction and de-bugging.

Engineers have since come up with a multitude of graphics and sound capabilities, as a result of the explosion of interest in arcade-type games on home computers. Each new model introduces more extensive features that are soon incorporated in professionally written software.

**Working Tools**
These are some of the tool kits and BASIC extension packages that are available for some of the most popular home computers. Packages for creating a sprite facility on computers that don't feature them as standard are becoming increasingly popular

| Tool Kits | |
|---|---|
| **SUPER TOOL KIT** | Available for the 16K and 48K Spectrum, from Nectarine |
| **SPECTRUM EXTENDED BASIC** | For the 48K Spectrum, from CP Software |
| **SPECTRUM KEYDEFINE** | For the 48K Spectrum, from Scientific Software |
| **PROGRAMMER'S AID** | For the Vic-20, from Commodore |
| **BUTI** | For the Vic-20, from Audiogenic |
| **TOOL BOX** | For the BBC Models A and B, from BBC Software |
| **SPRITE MAGIC** | Available for the Dragon 32, from Merlin Micro Systems |
| **SPRITE GRAPHICS** | For the 48K Spectrum, from B!Sides Software |
| **SPRITE MASTER** | For the BBC Model B, from Micro Dealer UK |

However, with one or two exceptions, the built-in BASIC provides little or no improvement on the earliest versions. This results in the user working out routines, often using repeated PEEKs and POKEs, to incorporate these new features into the range of available commands. As a consequence, there are now many utilities, tool kits and extensions to BASIC available for most of the popular machines. In general, these either give easier access to existing facilities (e.g. sprite or sound editors), extend software facilities (e.g. sprite creators), or provide simple aids to BASIC programming.

Extensions such as these can be located in RAM, internal ROM or on ROM cartridge. A ROM extension is preferable to one loaded into RAM, as it does not take up any user memory and is protected from inadvertent erasure. Generally, a program written with the aid of a tool kit will run only on another computer that is similarly equipped. However, there are utilities available that will generate free-standing programs, which will then run on an unexpanded version of the computer. This is the basis of most graphics and sprite editors, as well as some sound editors.

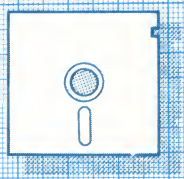
Useful features to look for in BASIC extensions are special graphics commands (such as PAINT, DRAW, PLOT, CIRCLE etc.) and sound commands (like SOUND, PLAY, MUSIC, ENVELOPE etc., or words that describe a sound effect, like BANG or ZAP). Other useful facilities are structured programming commands, such as REPEAT...UNTIL and IF...THEN...ELSE. Statements such as these enable the user to write programs that progress in a logical sequence, and avoid the untidy and difficult to understand code that results from an indiscriminate use of GOTO.

## Simon's BASIC

Currently, the most complete extension to the BASIC language is 'Simon's BASIC', which is available on the Commodore 64 in the form of a ROM cartridge. The standard Commodore BASIC, built into the 64, is rather antiquated, in that it provides a bare minimum of dedicated commands and no structured programming commands. Although it does have advanced hardware features, such as a comprehensive sound synthesiser, high resolution graphics and sprite graphics, BASIC control over these functions is via PEEK and POKE. Simon's BASIC provides a considerable extension to Commodore BASIC, by way of the following extra facilities:

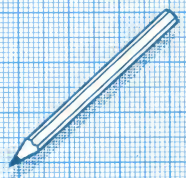
## Machine Tools

These commands are representative of the facilities that any good extension to BASIC will include



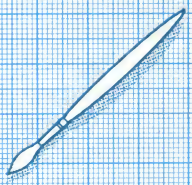
Various DISK utility commands will be offered for formatting new disks, copying and deleting individual files, and making complete back-up copies of a whole disk

DISK



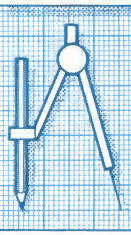
In addition to commands for drawing straight lines in high resolution graphics, there may well be a facility to DRAW from point to point, which is an easier way to build up images

DRAW

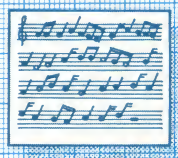


PAINT provides a means of filling up a defined area on the screen with any chosen colour. On some machines the cursor is positioned in the middle of the shape, and the computer then fills up the area by painting outwards from the centre until it encounters a solid line

PAINT



The purpose of graphics commands such as CIRCLE is obvious, but some will allow part circles (arcs) and ellipses to be drawn

CIRCLE

Though the sound functions vary greatly from machine to machine, a MUSIC command usually signifies the ability to play a sequence of notes, which have been previously defined in a string

MUSIC

The directory on a disk keeps an up to date list of the names, types and sizes of all files. A DIR command will usually let you view the directory on screen, without losing the program currently in RAM

DIR

Accessing the joysticks on many computers is a difficult task involving use of PEEK and POKE. JOYstick commands will place the position of the joystick directly into BASIC variables

JOY

KEVIN JONES

(1) A comprehensive set of programming aids, including functions that give extra control over program listing, de-bugging and security aids (protection against unauthorised copying of your programs).

(2) Additional string handling and text manipulation commands.

(3) Extra arithmetic operators and numeric conversion commands.

(4) Simplified disk handling commands.

(5) High resolution graphics commands that allow text to be mixed with point plotting and shape drawing. This includes a facility to colour in outlines.

(6) Low resolution graphics and screen handling commands that can duplicate assigned graphic areas and manipulate the screen area with ease. This also enables the contents of any screen to be saved on disk, tape or printer.

(7) Easy to use sprite creator and editor.

(8) Structured programming using procedure commands such as PROC, CALL and EXEC; plus loop and condition testing routines, like REPEAT... UNTIL, LOOP...EXIT, IF...END LOOP and IF...THEN ...ELSE, which generally eliminate the need for GOTOs and GOSUBs.

(9) Sound creation routines that allow the full range of the 64's sound capabilities to be accessed using simple sound shaping and playing commands.

(10) Simple light pen, joystick and paddle commands.

It is very unusual for an extension to include such a complete range of additional routines. Most packages supply utilities and commands for one specific programming area. For example, the Super Expander cartridge from Commodore for the Vic-20 provides a simple range of high resolution graphics and music commands only. The most popular extensions are those that include aids in program construction. These generally provide single key entry commands and various automatic routines that simplify line numbering, editing and de-bugging in the direct mode.

It is commonplace for utilities and extensions to allow the built-in capabilities of a computer to be accessed more easily. Routines that add significantly to a home computer's capabilities are harder to find, but ingenious packages are becoming available. For example, the many advantages of sprite graphics for fast action arcade-type games have inspired some enterprising companies to write sprite-creating utilities for computers that don't have this feature.

The BASIC utilities, tool kits and extensions we have outlined make up a small fraction of the improvements and aids available. Although the present tendency among manufacturers is towards providing comprehensive and advanced versions of BASIC, there will always be a need for software aids to help make programming a creative pleasure rather than a heavy chore.

# Voice Of Authority

**Speech recognition systems are being increasingly used in commercial and security applications. However, their powers are restricted by the computer's memory capacity**

For a computer to be of any use it must have a workable means of allowing commands and information to be fed into it. The 'interface' that we normally use to communicate with a home computer is a keyboard (though mice and joysticks are possible alternatives). By using a keyboard, however, we find that we are forced to communicate with the system by means of an artificial language. Commands such as CLS, DIRECTORY, RUN, LOAD and SAVE may be meaningful to the operating system but they aren't 'natural'.

The natural communication system for humans is speech, not typing messages on keyboards and watching the replies on television sets. If a computer could be made to understand spoken commands — even if they were phrased in the same way as the ones given through a keyboard — it would be far easier to use, especially by those with a physical handicap. Before any computer system can 'understand' spoken words, it must first process the sound input: the analogue signals must be analysed and turned into a digital form that the computer can deal with. Although it seems to be an easy thing to generate electronically, speech is a remarkably complex combination of sounds.

Dreams of instant and complete speech recognition (as typified by the computer HAL in *2001 — A Space Odyssey*) are unlikely to be fulfilled for many years yet, if ever. The voice input typewriter is equally distant; yet the technology for both this and the 'understanding' computer already exists. But neither is available at low cost, because there is a major difficulty in creating speech recognition systems: words can sound the same but have different meanings, depending on the context that they appear in. The processing power needed to solve this problem is simply not available at a reasonable price.

Although researchers have created systems that approach this goal, they have discovered that increasing the number of speakers who can be recognised by the computer has the effect of reducing the number of words that can be recognised at any one time. Typically, a multi-speaker recognition system will allow between 20 and 30 words to be recognised at a time, with a success rate of around 85 to 90 per cent.

The potential uses of speech recognition systems are considerable. The German Post Office uses one to assist with sorting mail; and there are now many applications in aerospace, both military and civil, where pilots have literally not enough hands and feet to control their aeroplanes. In all these situations the number of words that can be recognised at any one time is limited to around 20. However, this doesn't mean that the overall system is restricted. The user is selecting one of the 20 words from a 'menu', and each recognised command produces a further menu of words to choose from. Only when the complete sequence has been successfully recognised will any action be taken by the computer. In the case of the sorting office the first level of sort could be by state, and once the correct state is selected the next sort could

## Parts Of Speech

One technique of speech recognition simply involves digitising the signal and performing extensive 'pattern recognition' analysis. A more efficient method is to use hardware pre-processing, in which a number of independent circuits measure the signal for voiced sound (e.g. vowels), fricatives (s,f,t, etc), and short periods of silence (e.g. between syllables). The output from each of these filtering devices is a string of 1s and 0s, which the computer compares with a library of stored examples, selecting the nearest match as the word it recognises

KEVIN JONES

be by town, then by village, etc. Only at the lowest level would the item finally be sent on its way, thus ensuring the maximum reliability of the operation.

## Voice Analysis

Speech recognition is usually tackled in one of two ways. The 'quick' way is simply to feed all the speech through an analogue-to-digital convertor, and use the power of the computer to perform all the analysis. Unfortunately, this method has a number of drawbacks, most notably the time taken to perform the analysis. Systems using this method can take up to two or three seconds to recognise the input. For speech recognition to be of any real use the computer must 'understand' the speech as fast as another human, and the number crunching approach rarely achieves this.

The other method is to use pre-processing. Rather than analyse the speech signal mathematically, it is possible to do much of the work with standard electronics. What is then delivered to the computer is information about the spoken input: the frequency content, pitch, energy, etc. Frequencies can be measured by filtering the signal and detecting the level in each frequency band, rather like using tone controls on a hi-fi to 'bring out' the bass drum. Because all this electronic processing is done at the same time as the original speech signal is fed to the circuits, the analysis is almost instantaneous. Performing a similar operation on the digital data from an A/D convertor would require several computers working on the numbers at once. The pre-processing method is still at the research stage — no commercial system using it has yet been marketed — but it certainly appears to have more potential.

Once the information about frequency content, pitch, energy, etc. has been extracted from the original signal (regardless of the method), the actual recognition is performed by comparing the current set of figures with a number of models stored in the computer's memory. These models are created by 'training' the recognition system. The words that are to be recognised are spoken into the system one at a time and the resulting information is stored in a digital 'library' of examples. The complete set of words is then spoken again and the computer compares the input with its current model. If they agree, the second set of information is added to the first to form a more complete version of the model. This can be a continuous process, constantly adding new information to the library for more and more speakers.

To recognise a spoken word, the computer must match the pattern of information from the input with one or more of the models stored in the current library. In many cases, several possible matches will be found as parts of other words will match the input pattern. The first two syllables of 'international', for example, are the same as those of 'interpreter'. At the end of the search, one word should stand out as being more perfectly matched than any of the other possibilities, and this is the one that the computer will interpret the input as being.

IAN McKINNELL

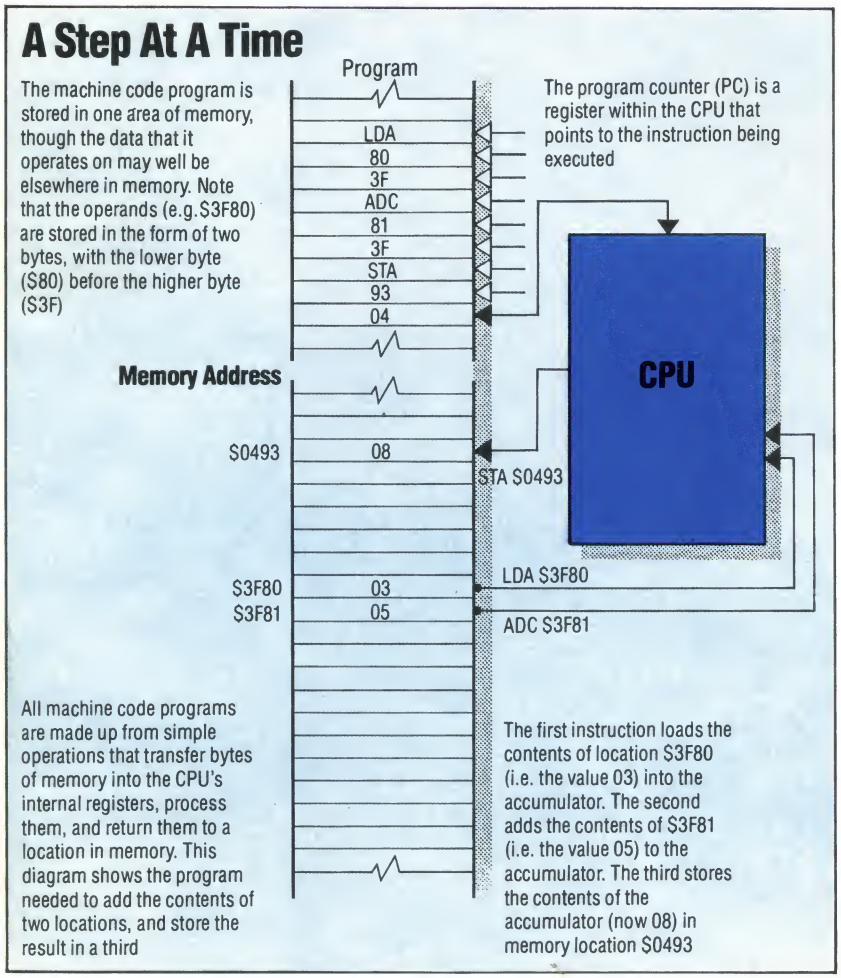
**Environmental Control**
Most recent applications of speech recognition are of an educational nature. One of these is called the 'limited environment', which involves a computer, a robot arm, and a number of simple objects that the arm can manipulate. Speaking into a microphone, the user can instruct the arm to 'PLACE THE EGG IN THE EGG CUP'. The computer will have to interpret the commands, and look up the positions of the objects in its memory

Speech recognition facilities are certain to find many applications in the future, but they are likely to be most readily used as a 'front-end' for complex software packages, such as databases, where the commands are selected from an on-screen menu. This type of application will remove the single biggest obstacle to computer usage by non-experts: the keyboard. Viewdata systems such as Prestel have reduced the input device to a simple numeric keypad, but this substantially limits the amount of interaction that a user can achieve. A speech-driven interface that can recognise a standard set of database interrogation commands, as well as numeric symbols and the letters of the alphabet, would provide a powerful facility that requires little, if any, conventional computer training to use.

There are now commercially available recognition units that can be plugged into home computers, but these are very unsophisticated devices. Systems like 'Big Ears' and Heuristic Inc's 'Speech Lab' use a lot of processing power to recognise just a few words spoken by one person. What is needed before speech recognition can become really useful is an ability to recognise words spoken by *any* person, regardless of dialect or accent. The limiting factor, at this stage, is the amount of memory available to hold the models. One interesting possibility is that of using a video disc to hold a standard set of models: this would use hardly any internal memory and the reduction in speed would be barely noticeable.

# Machine Code

**Learning machine code requires a considerable conceptual jump from Basic, but it offers a massive increase in speed and efficiency**


A Step At A Time
Program
The machine code program is stored in one area of memory, though the data that it operates on may well be elsewhere in memory. Note that the operands (e.g.S3F80) are stored in the form of two bytes, with the lower byte ($80) before the higher byte (S3F)
The program counter (PC) is a register within the CPU that points to the instruction being executed
LDA
80
3F
ADC
81
3F
STA
93
04
Memory Address
CPU
S0493
08
STA S0493
LDA S3F80
S3F80
03
S3F81
05
ADC S3F81
All machine code programs are made up from simple operations that transfer bytes of memory into the CPU's internal registers, process them, and return them to a location in memory. This diagram shows the program needed to add the contents of two locations, and store the result in a third
The first instruction loads the contents of location S3F80 (i.e. the value 03) into the accumulator. The second adds the contents of S3F81 (i.e. the value 05) to the accumulator. The third stores the contents of the accumulator (now 08) in memory location $0493

KEVIN JONES

So far in THE HOME COMPUTER COURSE, all our programming has been centred around the language BASIC, because it is both versatile and easy to use. However, as your experience grows, and the programming projects you tackle become more adventurous, it will not be long before you encounter the limitations of this language. You will soon find that graphics can't be moved around the screen as fast as you would like, and that you often have to resort to the confusing PEEK and POKE commands to make the best use of your machine's facilities.

By contrast, programming in machine code imposes very few constraints on what you can do, and compared with BASIC, gives the impression of almost infinite speed. However, comparatively few home computer owners make the jump from BASIC to machine code, partly because using machine code is a far more labour-intensive programming process, and also because it is conceptually quite different from BASIC or any other high level language. Nevertheless, it is extremely worthwhile to have an understanding of machine code; and in this article, the first of two, we look at the fundamental procedures involved in using it.

Machine code, as we have explained before, is the language understood by the microprocessor (the CPU) that forms the heart of your computer. This microprocessor can only perform very simple functions (it can add two digits of a number, for example, but can't multiply them). It does, however, perform these functions at very high speeds. Every operation of a microprocessor is specified in terms of the number of 'clock cycles' taken. If the CPU in your computer runs at 1 MHz, then a clock cycle is one microsecond, and an operation that takes four 'clock cycles' to perform does so in four millionths of a second.

As a consequence, any program written in machine code will consist of a large number of instructions, and any function must be built up 'by hand' from simple operations. All machine code programming consists of manipulating individual bits or bytes of memory, using simple logic functions like AND, OR and NOT, and elementary binary arithmetic.

This is one reason why writing machine code is a slow task; the other is that it is the programmer's responsibility to know where everything is kept in memory. In BASIC, whenever a statement like LET A=5 is encountered, it is the job of the BASIC interpreter to find a space in memory to store that variable. Furthermore, whenever A is referenced later in the program, it will remember where to look for the necessary data. When you first start programming in machine code you discover that you have to specify an address (a memory location) for every piece of data you need to store, and it is up to you to ensure that it is not accidentally overwritten with other pieces of data.

Let's look at what machine code consists of. (Incidentally, all our examples will refer to eight-bit CPUs, such as the Z80 and 6502; 16-bit devices work in a similar manner but process twice as many bits with each operation). The microprocessor is connected to the computer's memory by two busses (a bus is merely a group of wires or lines): the address bus and the data bus (see page 144). There is also something called the control bus, but this provides timing signals for the CPU and is not used by the programmer.

The address bus is 16 bits wide, and by placing a pattern of bits on this bus, the CPU can select any of the 65,536 bytes in its 'memory map' (see page 329). In a typical home computer, some of these locations will consist of RAM, some of ROM, some of special input/output chips, and some will be unused. If the CPU wants to read a memory location (one of the lines in the control bus

indicates whether a read or write is to be performed), then the selected byte will place its contents on the data bus, in the form of a pattern of eight bits. Similarly, the CPU can write a pattern of eight bits into any chosen location. The CPU has no knowledge of which parts of memory are ROM and RAM, so getting the addresses right is another crucial responsibility of the programmer.

Inside the microprocessor, there are perhaps half a dozen 'registers', which are like individual memory locations and are used for storing temporary results and performing the logic and binary arithmetic functions. Most of these registers are equivalent to one byte of memory, though some are 16 bits wide. One of the latter type is called the Program Counter (PC) register, and this contains the address in memory of the machine code instruction that is currently being performed. You can think of this as being similar to the line number in a BASIC program.

Another of the most important registers (but this time just eight bits wide) is the 'accumulator'. As the name suggests, this register can accumulate totals (that is to say, bytes can be added to it or subtracted from it), and indeed this is usually the only register that can perform any kind of arithmetic. So, a very simple machine code program might be specified as follows:

1) Load the accumulator with the contents of memory location $3F80. Addresses in machine code are usually written in hexadecimal (see page 179). Hexadecimal numbers are indicated in writing by prefixing a special sign, usually a $.

2) Add to the accumulator the contents of memory location $3F81, allowing for the fact that the result may be larger than can be stored in a single byte — in which case there will be a 'carry bit' as well.

3) Store the new contents of the accumulator (i.e. the result) in memory location $0493.

Each of these constitutes a machine code instruction, and the program would normally be written thus:

```
LDA $3F80   (LoaD Accumulator)
ADC $3F81   (ADd with Carry)
STA $0493   (STore Accumulator)
```

The comments in brackets, like BASIC REMark statements, have no effect. The first entry on each line is called the 'opcode', and this indicates the nature of the operation. The second column contains the 'operand' — the details of, or whereabouts of, the data that is to be operated on. A microprocessor will usually feature several dozen possible opcodes (that is to say, it can perform several dozen types of simple operation), and each opcode will occupy just one byte of memory when it has been entered into the machine.

An opcode can therefore by specified as a number in the range 0-255 (or, more properly, in the hex range $00 to $FF). However, while a program is being developed, it is more usual to make the listing more readable by using three letter mnemonics, such as LDA, ADC and STA.

**Flashing Lights**
The idea for the huge panels of lights often seen on computers in films came from the 'front panel' found on many mini-computers. This front panel was a line of lights and switches representing the CPU's address and data buses. Before keyboards were interfaced, all machine code programs had to be entered in binary in this form

KEVIN JONES

Each of the three operands shown consists of a hex number in the range $0000 to $FFFF, and uses up two bytes of program memory space. However, some operands are just one byte long, and some opcodes don't have operands at all. The short program that we have given would therefore occupy a total of only nine bytes — not including the three memory locations ($3F80, $3F81, and $0493) that the program will operate on. For this trivial exercise, the following BASIC program would achieve exactly the same effect, but would occupy nearly 50 bytes and perform the operation at least a hundred times slower, because of all the time taken by the interpreter to translate it:

```
10 A = PEEK (16256)
20 A = A + PEEK (16257)
30 POKE 1171,A
```

N.B. The locations used by this particular program may not be suitable for your machine.

In the next instalment of THE HOME COMPUTER COURSE, we'll look at how machine code is entered into a home computer and run, and the different ways in which machine code is expressed.

**LDA — LoaD Accumulator**
Transfers the contents of a single memory location (byte) into the internal accumulator register

**STA — STore Accumulator**
Performs the opposite process to LDA

**ADC — ADd with Carry**
Adds the contents of a memory location to the current contents of the accumulator, creating a carry bit if necessary

**SBC — SuBtract with Carry**
This is the inverse function of ADC

**JMP — JuMP**
Transfers program operation to a new location. This is similar in operation to a BASIC GOTO statement

**Opcodes**
These are just a few of the opcodes (types of operation or instruction) that a typical microprocessor can execute

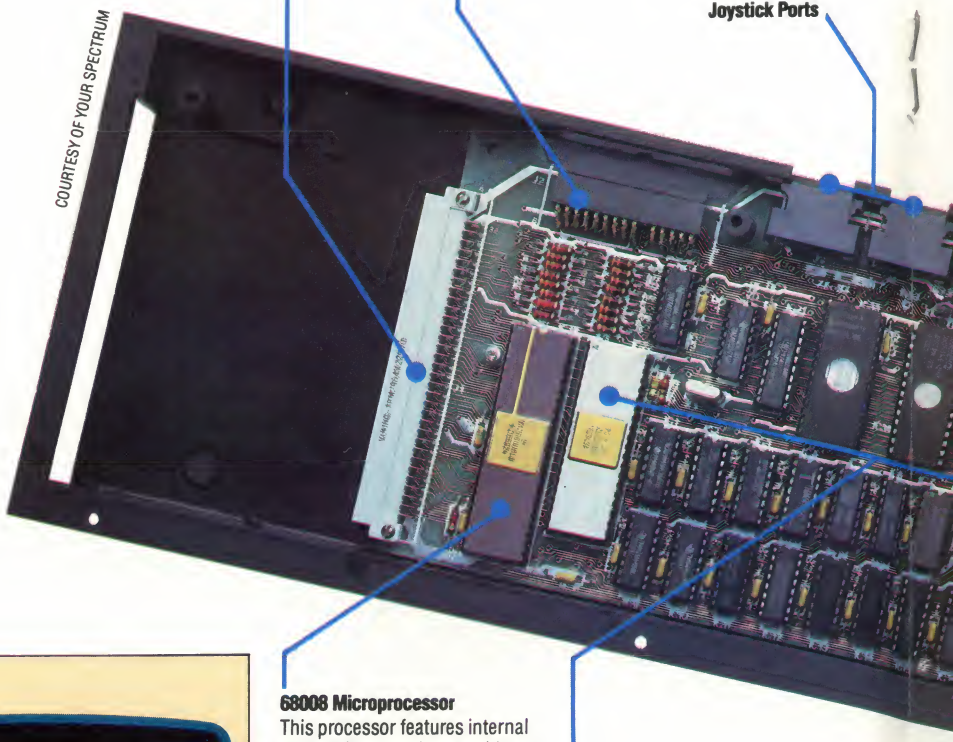
# Sinclair QL

## The Quantum Leap offers the most advanced microprocessor on any home computer, and the potential for half a megabyte of memory

All Sir Clive Sinclair's innovations in the field of home computers have represented quantum leaps both in terms of technology and value for money, but his latest microcomputer is the first of his machines to take that description as its name: the Sinclair Quantum Leap (QL). At £399, it is aimed at a growing number of users who are either serious computer enthusiasts or have business as well as home applications in mind. As such, it represents very serious competition for machines like the Commodore 64 and BBC Model B, though in terms of technical specification it is dramatically superior.

It is quite apparent that the QL has been designed by stringing together all the components and features that currently represent the height of computer fashion. Making a break from the usual choice of Z80 or 6502, the CPU is a member of the Motorola 68000 family, which is currently the most sophisticated microprocessor found in any microcomputer and used in machines like Apple's Lisa (see page 261). However, the CPU is a 68008, which means that though its internal registers are 16-bit (and it can perform many functions across a full 32 bits), its external data bus is only eight bits wide. This will slow the operation of the CPU very slightly, because the loading and storing of the registers will have to be done in halves. But this also means that the cost of the memory chips is kept down, and economics is often a prime consideration in Sinclair's choice of components.

The QL comes with 128 Kbytes of RAM as standard, but will be expandable to 512 Kbytes (or 'half a meg', as it is termed) with future add-ons. This large memory is particularly useful for business applications, as it reduces the frequency with which the program must refer to off-line storage. This storage consists of two Microdrives built into the casing, offering around 100 Kbytes each. Though this does make the QL a self-




**Expansion Interface**
Peripherals, and up to 0.5 Megabytes of RAM can be coupled on here

**ROM Cartridge Slot**
Up to 32K of additional ROM can be plugged in here

**Joystick Ports**

**68008 Microprocessor**
This processor features internal 16- and 32-bit registers, with an 8-bit external data bus

**Custom Chips**
An increasing number of new computers feature a custom-designed chip. The QL has two, to handle the display and various interfaces

## QL Software

QL Quill is a word processing package that displays the text on the screen in the same format as it will be printed

QL Abacus is a spreadsheet with the unusual feature that cells can be referred to by name instead of just co-ordinates

QL Archive is a database package. Record layouts can be designed by the user, with the aid of a screen editor

QL Easel is a graphics utility designed for producing graphs and charts, handling aspects of design like scaling automatically

IAN McKINNELL

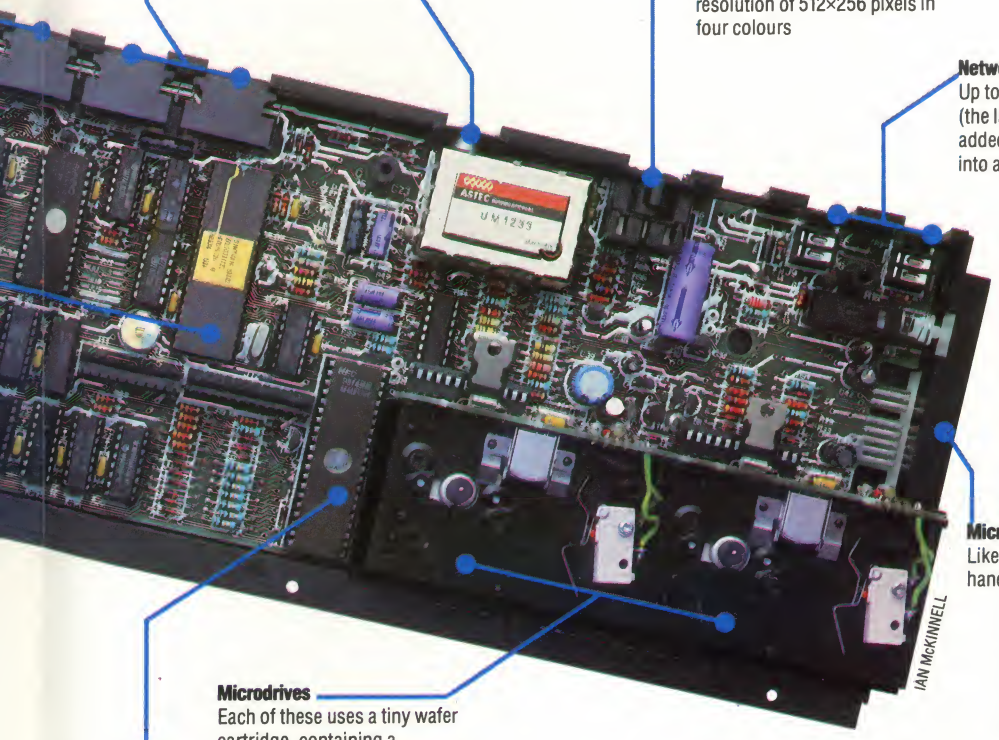
**Keyboard**
Though the keyboard is based on a membrane construction (thereby guarding it against coffee spillages, etc.) it features 65 full-travel keys, and the 'feel' is every bit as good as some of the most expensive business machines. There are four cursor control keys, and five programmable function keys. The copyright symbol and pound sign are also included

**Serial Ports**
Two RS232 ports are incorporated, suitable for driving a printer and a modem. The more common (Centronics) printer interface must be purchased as an add-on

**TV Socket**
The QL will work with a TV set, but will normally only display 40 or 60 columns, where 85 are possible with a monitor

**Monitor Port**
Unlike the Spectrum, the QL can drive an RGB monitor directly, and indeed this is needed to take advantage of the maximum resolution of 512×256 pixels in four colours

**Network Interface**
Up to 64 QLs and Spectrums (the latter with Interface 1 added) can be linked together into a Local Area Network



IAN McKINNELL

**Microdrive Extension Slot**
Like the Spectrum, the QL can handle up to eight microdrives

**Microdrives**
Each of these uses a tiny wafer cartridge, containing a continuous loop of tape to store up to 100K each

**Second Microprocessor**
This Intel 8049 controls the keyboard, sound, and serial ports, leaving the 68008 free for running user programs

**PROTOTYPE BOARD**

**The photograph shows the PCB of a pre-production QL, so elements of the design may change for the production models**

## Sinclair QL

**PRICE**
£399 inc VAT

**SIZE**
472×138×46mm

**CPU**
Motorola 68008

**CLOCK SPEED**
7.5 MHz

**MEMORY**
128K RAM, expandable to 512K
32K ROM, expandable to 64K

**VIDEO DISPLAY**
25 lines of 85 characters (with monitor), high resolution graphics: 512×256 pixels (4 colours), 256×256 (8 colours)

**INTERFACES**
Serial RS232 (2), Joysticks (2), Microdrives, LAN, TV, RGB monitor

**LANGUAGE SUPPLIED**
BASIC

**OTHER LANGUAGES AVAILABLE**
Several are planned, most notably the 'C' language

**COMES WITH**
Instruction manual, four applications programs

**DOCUMENTATION**
The provisional manual is of a high standard, comes in a ringbound folder, and includes manuals for the standard software

contained business system, the Microdrives must be viewed as something of a weak point when compared with the remarkably efficient processor. It takes an average of 3.5 seconds to locate a data item on the Microdrive, compared with perhaps half a second for the new generation of mini floppy drives.

Sinclair say that they intend to produce an interface to a hard (Winchester) disk unit, but there are no plans for floppy disks — though some independent manufacturer will undoubtedly offer them. Without disks, unfortunately, the QL will be unable to run the Unix operating system, which is usually considered to be one of the main reasons for choosing a Motorola 68000 CPU, and is tipped to replace CP/M as the standard operating system for business software.

The QL comes with four business packages as standard, all developed by the software house Psion. Quill is a word processor; Abacus, a spreadsheet package; Archive, a database; and Easel a graphics package. All run under the resident operating system, which Sinclair have dubbed QDOS. The popularity that this machine is likely to achieve means that a lot of software will be developed for it, though it will not be easy for software houses to transfer their existing packages onto the QL. Still, it might be argued that if Sinclair adopted industry standards, its products would not have the market lead that they do have.

The resident BASIC has been upgraded from the Spectrum version, and as if the name Quantum Leap weren't immodest enough, Sinclair have called this SuperBASIC. It includes facilities for handling procedures (thereby encouraging structured programming) and for accessing the operating system from within a BASIC program. Both the BASIC and QDOS are contained in the 32 Kbytes of ROM as standard.

The Sinclair QL is without doubt a very impressive machine and, perhaps more important, has sufficient expansion possibilities to guard against obsolescence. It is a fitting addition to a long line of Sinclair milestones: the ZX80, the ZX81, and the Spectrum.

# Sound Principles

## The sound functions of the Atari models include four independent voices

The Atari sound facilities are good — as can be heard in many of the cartridge games — though the means of controlling them are a little idiosyncratic. Four independent square wave oscillators are provided, each with a range of three octaves. As a bonus, the oscillator output can be distorted in seven ways to colour the sound. These facilities are easily accessible from BASIC via the SOUND command provided, but this doesn't make full use of the extra features of the Atari sound chip POKEY, which with high-pass filters and special modes of operation can extensively modify the sound produced. As a consequence, the full range of sound control can be fully exploited only by using complex POKEs or machine code, which is beyond the scope of this part of the course. Output is via the television speaker only.

### SOUND

This is a very simple command with the following format:

SOUND O,P,D,V

O = Oscillator (0-3)
P = Pitch (0-255)
D = Distortion (even numbers 0-14)
V = Volume (1-15)

Each SOUND command can select only one oscillator, so it is impossible to start more than one oscillator at a time. This is not normally a problem, but if music is programmed using all oscillators for four-part harmonies the delay is noticeable.

Pitch is calculated a little strangely and as a consequence some frequencies are inaccurate. Frequency decreases as the pitch number increases, giving an effective range from 'C' at 29 (1046.5Hz) to 'C' at 243 (130.81Hz). The following table indicates some of the pitch numbers for music note symbols. A full list is given in the Atari BASIC reference manual.

| | Octave-1 | Octave-3 |
|---|---|---|
| (Mid) | C — 121 | C — 29 |
| | B — 128 | B — 31 |
| | A — 144 | A — 35 |
| | G — 162 | G — 40 |
| | F — 182 | F — 45 |
| | E — 193 | E — 47 |
| | D — 217 | D — 53 |
| | C — 243 | C — 60 |

The distortion parameter 'P' is equivalent to the noise channel on most computers but it is far more versatile. Each even number causes a different arrangement of random pulses to be mixed with the standard oscillator output. Curiously, 10 gives

# Light Refreshment

## A quick look at the Oric's graphics shows many similarities with the Spectrum

The Oric-1 home computer was released in the middle of 1983 and is designed as an obvious rival to Sinclair's ZX Spectrum. The Oric offers four modes of display. Only one mode, however, enables the use of high resolution graphics. There are eight colours available; foreground and background colours respectively being set by the commands INK and PAPER. Oric BASIC includes several special high resolution commands to aid the graphics programmer.

The screen is made up of 28 lines, each containing 40 character spaces. The Oric's characters are not designed using the usual eight by eight pixel grid, but are constructed on an eight by six grid. In high resolution mode the screen has 240×200 pixel resolution, the bottom three lines being reserved for information such as error messages. There is no PAINT-type command, but with a little thought it is possible to accomplish the function using the FILL command. As with the Spectrum (see page 392) it is possible to mix high resolution graphics and text together on the same screen, but the Oric allows each line inside a character square to be coloured individually, whereas the Spectrum allows only one colour to be specified within any one character square.

Now let us look in more detail at the low resolution modes offered by the Oric-1. The Oric has three low resolution modes: TEXT, LORES0, and LORES1. The only difference between LORES0 and LORES1 is that they use different character sets. In the TEXT mode, letters can be positioned horizontally by the TAB command. In the two LORES modes, however, this facility is improved to

a distortion-free signal, not 0, as might be expected. With experimentation the careful use of distorted sounds can provide interesting tones and is particularly useful for special effects.

Volume 'V' can be set between 1 and 15 and a reasonable average level would be 7 or 8. Note that there is no convenient way of timing the duration of notes or the pauses between them. The usual method under these circumstances is to use carefully timed FOR...TO...NEXT loops.

To illustrate the use of SOUND, the following commands play an undistorted 'G' in octave 3 on oscillator 1 at a volume of 8 for 50 FOR...TO...NEXT steps:

```
10 SOUND 1,40,10,8
20 FOR N=1TO50:NEXT N
30 END
```

The END in line 30 turns all oscillators off. Alternatively, a new SOUND command for the same oscillator stops the old note and immediately plays the new one. A program to play a simple tune could be constructed like this:

```
10 REM*DIXIE*
20 FOR I=1TO7
30 READ N:REM*NOTE*
40 SOUND 3,N,10,7:REM*PLAY NOTE*
50 FOR P=1TO400:NEXT P:REM*PAUSE*
60 NEXT I
70 DATA 219,162,128,144
80 DATA 162,193,162
90 END
```

It is possible to access the sound capabilities of the Atari's POKEY chip from BASIC by POKEing numbers into memory locations 53760 to 53763. With this method, sound routines run faster and all oscillators can be started at one time. All information necessary to accomplish this, plus more adventurous machine code techniques, are contained in *De Re Atari*, available from the Atari Program Exchange (APX), and also the excellent *Atari Sound And Graphics*, published by John Wiley & Son.

allow the user to specify vertical as well as horizontal positions using the PLOTx,y,A$ command, where x and y are the co-ordinates of a particular character position and A$ is the word or phrase to be PRINTed. The following short program demonstrates how this facility may be used to write a name vertically.

```
10 REM VERTICAL LETTERS
20 CLS
30 LORES0
40 A$="STEVE"
50 FOR X=1TO5
60 B$=MID$(A$,X,1)
70 PLOT16,11+X,B$
80 NEXT X
90 END
```

The command HIRES allows the user to enter the Oric's high resolution mode. In HIRES mode the screen has its origin in the top left-hand corner of the screen.

There are several commands in Oric BASIC to help specifically with graphics: CURSETx,y,k positions the cursor at the point with co-ordinates (x,y). The third number 'k' allows different functions to be employed with CURSET.

| Value Of k | Function |
|---|---|
| 0 | plots pixel in background colour |
| 1 | plots pixel in foreground colour |
| 2 | inverts colours |
| 3 | does nothing |

CURMOVx,y,k is similar to CURSET, except that the cursor movement is relative to its previous position. DRAWx,y,k draws a straight line from the current cursor position to a point x units across and y units up. CIRCLEr,k is a command that will draw a circle of radius 'r' on the screen. PATTERNn is an unusual and interesting command. PATTERN breaks up lines or circles drawn into a series of dots or dashes. The exact pattern is defined by the number 'n', which lies in the range 0 to 255. The Oric takes this number and uses the bit pattern of its binary equivalent to produce a repeating pattern of dots, dashes or spaces. Here are two examples to illustrate its use:

| Value Of n | Binary Equivalent | Pattern Produced |
|---|---|---|
| 170 | 10101010 | - - - - - - - - |
| 15 | 00001111 | — — |

Finally, there is the command FILLa,b,n. Each row of every character space on the Oric screen has a number associated with it that relates to the foreground and background colours, the character present and whether the character is flashing or not. This number is known as the 'attribute' of that row. FILLa,b,n fills 'b' character cells by 'a' rows with the attributes represented by the number 'n'.

```
10 REM CONE
20 HIRES
30 CURSET120,0,3
40 PAPER3 : INK4
50 FOR R=1TO65
60 PATTERN 200-R
70 CURMOV0,2,3
80 CIRCLE R,1
90 NEXT R
100 END
```

**Cone PATTERN**
This program demonstrates some of the high resolution capabilities of the Oric-1. A cone shape is drawn using a set of circles of increasing radius. Note also the use of the PATTERN command to break up the circumference of the circles as they are drawn

# Dpef Dsbdljoh

## Encryption was one of the earliest applications of computers. Nowadays, devising and cracking a simple code is within the powers of the Basic programmer

All our communications with others are codified. Whether we use speech or written language, both are presented in such a way as to be intelligible only if the person receiving the message can interpret the code. The same is true of our conversations with computers. Most home computers rely on a dialect of BASIC in order to be accessible to people, but we know that the machine itself does not use this language to perform its functions: it must first interpret the BASIC statements into a purely numerical form that it can then use to set up the switching sequences defined in the program, and thus produce the desired results. Codes of this sort — human and programming languages — are easily accessible in our everyday lives. Anyone can learn French, German, BASIC or FORTRAN, given the effort and the will.

### Data Compression

Computer users who need to store large numbers of text files are constantly searching for ways to compress the data in those files. One way to achieve this is by tokenisation. In much the same way as Sinclair's ZX series of microcomputers produce a whole BASIC reserved word at the touch of a single key, a token can be substituted for a commonly-used word or phrase.

In addition, encoding techniques are also used to compress the data even further. Compact, a Unix utility, is generally reckoned to compress text files by an average of 38 per cent, and Clip, which runs under CP/M, regularly achieves even better results. Compactor, which runs on the Commodore 64, performs the same function for BASIC programs by removing REMs, unnecessary spaces and so on

But there is another type of encoding (more accurately called 'encryption') that has the very opposite of communication as its objective: its purpose is to deny understanding to all but the small group for whom the communication is intended. Until the second half of the 20th century, the transmission of information in a form not generally intelligible was restricted to governments and a few large industrial interests. But more recently, with the ever increasing use of vulnerable public telephone lines for the exchange of information, most of it with some commercial value, the practice of encryption has become more commonplace.

Cyphers and codes range from the very simple — the addition or subtraction of a given value to every byte, perhaps, or the formatted substitution of one character for another wherever it occurs — to the immensely complex cyphers that are being worked on in the most recent advances in number theory. These cyphers contain no element of repetition whatsoever, and hence are not vulnerable to frequency analysis decoding methods.

The simplest of all meaningful encryption techniques is perhaps Caesar's Cypher (which was probably first used at the time of the Roman Empire). The decryption of Caesar's Cypher requires only the message and a knowledge of the key, so there are no bulky code books or documents to be concealed, and no sophisticated machines required. Here is a simple message encrypted in Caesar's Cypher:

FMKC AMKNSRCP AMSPQC

We can make a few assumptions about these encoded words because of the way in which the encyphered groups are spaced out (though, of course, this could be intended to create confusion!). The most obvious thing that strikes us is that the message consists of three words: the first has four letters, the second has eight and the last has six. We can also assume that the second and third words begin with the same letter, and that the first and last words end with the same letter. The common ending letter here (C) is also one of the two letters in the message with the highest frequency (the other is M). This observation is of considerable value to the cryptanalyst — at least, as long as he knows which language he is working with. In English, the letter that occurs most frequently is E, followed by T.

With a sample as small as the one we have here (a total of only 17 letters, which any statistician will tell you is an insufficiently large sample upon which to base any analysis), our results are likely to be fallible. But let's try frequency substitution anyway, and see if the results are meaningful. Let's substitute the E for the C first:

FMKe AMKNSReP AMSPQe

The message is still meaningless, but there are other clues. What about the relationship between the original letter and the one we substituted for it? C is two places in front of E in the alphabet. What happens if we put the rest of the message through the same transformation? Two places behind M (our other most commonly occurring letter) is O, so let's try adding that piece of information:

FoKe AoKNSReP AoSPQe

In the first word we now have: '(something) vowel (something) vowel', which is a valid English construction. Furthermore, the final vowel is E, which is a common occurrence in English, so

perhaps we're on the right track. Let's put the rest of the message through the transformation. Two behind F is H; two behind K is M; and so our first word could be HOME . . .

Caesar's Cypher, then, is a substitution code that relies on 'sliding' the alphabet up or down a given number of places to determine the new value of each character. It can be refined further by using a string of key transformations — 24225, for example. In this case the first letter would be shifted two places, the second four, the third two, and so on. When the end of the key string is reached, we loop back to the beginning again. Using this key string, our sample message would be:

FKKC XMINSOCN AMPPOC

In this instance, frequency analysis will be entirely useless because there is no uniformity to the substitution — a letter will have different substitutes depending upon its position in the overall message. Another simple self-contained cypher renders the same message thus:

H PRUOECMUE OREMOTCS

If we look closely, we can see that this string of characters is in fact an anagram of HOME COMPUTER COURSE, complete with the two spaces between the words. Here, we are simply trying to determine the encrypting algorithm, given samples of both plain text and encrypted text — a surprisingly common procedure. If the cypher is to be understandable by the recipient of the message, then the jumbling up of the letters must be in some way predictable. This particular cypher, known as the Bar Fence for reasons that will soon become obvious, also requires the decoder to know the key — in this case it is 3. Let us take the first five characters and write them out with three spaces between:

H*** ***P***R***U

Recognise anything? Well try this then: write out the plain text message on three lines, going down and up between the lines, thus:

H * P R U
O E C M U E * O R E
M O T C S

The asterisks represent the spaces between words, and the method of encryption is plain.

The examples that we have cited so far have all been cyphers — defined as a method of secret writing using substitution or transformation of letters according to a key. Codes are rather different in that they tend to substitute whole blocks for other, normally smaller, blocks (thus allowing data compression at the same time). Their drawback is that they require both parties to possess a code book before messages can be communicated. One example of this technique uses a commonly available novel, newspaper or other piece of text and indicates the words that go to make up the message by simply giving the sequence number in which they occur. A piece of text like:

'Johnny went home and asked his mother if he might play a computer game. "Of course!", said his mother.'

could be the key to the code 3,10,3. Perhaps you can deduce the message . . .

A computer of any type is of tremendous value when attempting to either encrypt or decrypt messages in cypher. A prime requirement of Caesar's Cypher, for example, is the ability to move through an alphanumeric string, adding or subtracting a constant to the ASCII value of each character, which can then be printed. That constant must be capable of amendment when the program is run, and should make the alphabet wrap around (that is, looking up A where the key is one, should give Z). Thus:

PDWPO WHH BKHGO

## Caesar's Cypher

This program (written in Commodore BASIC) will encode text into Caesar's Cypher using a five element multi-key string. The message appears in plain text as it is being entered, and when RETURN is pressed the encrypted version is printed. The message should be entered without spaces or punctuation

```
10 INPUT "ENTER A FIVE FIGURE KEY";K$
20 INPUT "ENTER THE MESSAGE";M$
30 FOR I=1 TO LEN(M$)
40 LET J=I — INT(I/5)*5+1
50 REM *** ROTATES THROUGH KEY
60 LET M=ASC(MID$(M$,I,1)) — VAL(MID$(K$,
   J,1))
70 IF M<65 THEN LET M=M+26
80 PRINT CHR$(M);
90 NEXT I
```

For the Spectrum, line 60 should be replaced with:

```
60 LET M=CODE(M$(I)) — VAL(K$(J))
```

**Cryptanalysis**

One of the earliest uses of computers was to crack the very complex multi-key substitution codes in use by both sides during the Second World War. The Germans had developed a machine called ENIGMA that generated its own cyphers. The immensely complicated cryptograms that resulted caused the Allies to devote a great deal of effort to their interpretation. Success finally came to the Colossus group, working at Bletchley Park, of which Alan Turing was a prominent member

COURTESY OF THE SCIENCE MUSEUM

# A Matter Of Style

**Now that we have covered the fundamental rules of Basic, we can concentrate on important aspects of programming style and some new commands to perfect programming technique**

The computerised address book program we have developed in previous instalments of the course uses many of the more important features of the BASIC language, but certainly not all of them. In the concluding parts of the Basic Programming course we will look at where BASIC can take you next if you wish to become an advanced programmer. Unfortunately, this cannot be exhaustive, and readers are advised to refer to the owner's manual, or one of the many supplementary books that have been published for most of the popular home computers, for more extensive analysis of their machine's version of BASIC.

## Machine Language Programs

Most versions of BASIC allow routines written in machine language to be included as part of the program. Broadly, there are two ways of doing this. The simplest is to use PEEK and POKE. PEEK is a statement used to examine specific memory addresses. For example, LET X = PEEK(1000) will get the value stored in address location 1000 and assign it to the variable X. Executing PRINT X will then print the value that was (and still is) in location 1000. Here is a short program that will PEEK at the contents of 16 memory locations and print them out on the screen:

```
10 INPUT "ENTER 'PEEK' START ADDRESS";S
20 PRINT
30 FOR L = 1 TO 16
40 LET A = PEEK(S)
50 PRINT "LOCATION ";S;" CONTAINS: ";A
60 LET S = S + 1
70 NEXT L
80 PRINT "PRESS SPACE BAR TO EXAMINE NEXT 16
    LOCATIONS"
90 PRINT "OR RETURN TO END"
100 FOR I = 1 TO 1
110 LET C$ = INKEY$
120 IF C$ < > CHR$(13) AND C$ < > " " THEN
    I = 0
130 NEXT I
140 IF C$ = CHR$(13) THEN GOTO 160
150 GOTO 30
160 END
```

The loop in lines 100 to 130 checks the input from the keyboard and then either goes to the end of the program, if the character typed was a RETURN (13 in ASCII), or back to the beginning, skipping the INPUT statement.

If desired, the ASCII character of the memory location can also be printed by using PRINT CHR$(A). But be careful, as ASCII values lower than decimal 32 (ASCII for the 'space' character) are not uniformly defined. All ASCII values from 0 to 31 represent non-printable characters or special functions, such as cursor controls. About the only agreement between different computer manufacturers is that ASCII 13 is usually the carriage return and ASCII 7 sounds the internal speaker or produces a 'beep'.

POKE is the converse of PEEK. It allows you to write any value from 0 up to 255 in any RAM memory location. This facility must be used with extreme caution, however, as writing to a part of memory that is already being used by the program can cause unexpected or catastrophic results. Routines written in machine code can be POKEd to the appropriate addresses and invoked when the program is run by the CALL statement. How to write programs in machine code is beyond the scope of a course on BASIC. Suffice it to say that machine code runs very much more quickly than even the best BASIC dialects. In situations where speed of execution is essential, or where great precision is required, machine code is by far the better alternative.

## Moving The Cursor

Many home computers now allow locations on the screen to be addressed directly, but even if your machine does not support this, it is possible to move the cursor to the left, right, up and down the screen relatively easily. First you need to know what ASCII codes are used to represent the cursor control keys. The following short program will ask you to type a key and will then report the ASCII value corresponding to that key:

```
1 REM FINDING THE ASCII CODES FOR THE CURSOR
  KEYS
10 PRINT "PRESS A KEY";
20 FOR I = 1 TO 1
30 LET K$ = INKEY$
40 IF K$ = "" THEN I = 0
50 NEXT I
60 PRINT ASC(K$)
70 GOTO 10
80 END
```

This routine will also allow you to find the code for the RETURN key (usually 13), ESCape (usually 27) and the space key (usually 32), in addition to the codes for the cursor control keys. The Sord M23 computer, on which the programs in the Basic Programming course were developed, uses the

values 8 for cursor left, 28 for cursor right, 29 for cursor up and 30 for cursor down. Your computer will probably use different values. Substituting the values you have found for your computer's cursor control codes in the program above, try the following program:

```
10 PRINT CHR$(12): REM USE CLS OR
   APPROPRIATE CODE
20 FOR L = 1 TO 39
30 PRINT "*";
40 NEXT L
50 FOR L = 1 TO 22
60 PRINT CHR$(8); :REM USE 'CURSOR LEFT' CODE
70 NEXT L
80 FOR L = 1 TO 4
90 PRINT "@";
100 NEXT L
110 END
```

This should print a line on the screen looking like:

```
******************@@@@******************
```

Lines 20 to 40 would simply have printed a line of 39 stars. However, lines 50 to 70 'printed' the cursor left 'character' 22 times, so the cursor moved back along the line 22 places. Lines 80 to 100 then printed @ four times and the program then ended. Programming techniques such as this allow the programmer to move the cursor around the screen to print new characters in new positions that may not be known until the values are calculated in the program. This technique has the advantage of enabling ordinary screen characters to be used to plot simple graphs, without resorting to the computer's special graphics facilities (if it has any).

To see how this kind of cursor control can be used to produce graphs as an output from your programs, try the following short program:

```
10 PRINT "THIS PROGRAM PRINTS A BAR GRAPH OF
   3 VARIABLES"
20 INPUT "INPUT THE THREE VALUES "; X,Y,Z
30 PRINT
40 FOR L = 1 TO 2
50 FOR A = 1 TO X
60 PRINT "*";
70 NEXT A
80 PRINT CHR$(13)
90 NEXT L
100 FOR L = 1 TO 2
110 FOR A = 1 TO Y
120 PRINT "+";
130 NEXT A
140 PRINT CHR$(13)
150 NEXT L
160 FOR L = 1 TO 2
170 FOR A = 1 TO Z
180 PRINT "#";
190 NEXT A
200 PRINT CHR$(13)
210 NEXT L
220 PRINT
230 END
```

The program prints out a bar graph of the three variables. The bars are printed in horizontal rows, starting from the left and following the 'natural' cursor movement. Notice that a PRINT CHR$(13) is needed in lines 80, 140 and 200. They are needed because semi-colons at the end of PRINT statements suppress carriage returns (13 is the ASCII code for <CR>).

## More About Variables

So far we have treated variables as though there were only two kinds (numeric and string). In fact, there are several types of numeric variables recognised by BASIC, and a good programmer will always specify the right type to economise on memory and ensure correctness.

When a variable is declared in a programming language, a certain amount of memory space will be automatically allocated to store that variable. If the program knows that the variable will always be an integer, (e.g. LET SCORE = TOTAL + BONUS — PENALTY) less memory needs to be set aside for the variable. If we have a variable that can take an infinite number of different values (e.g. LET AREA = PI * RADIUS * RADIUS), more memory space will have to be allocated.

In the development of our computerised address book, we became familiar with the convention of specifying string variables by using the $ sign after the variable name (e.g. LET SCHKEY$ = MODFLD$(SIZE)). Variables without the 'dollar' sign were assumed to be ordinary numeric variables. However, similar conventions can be used after variable names to specify the type of numeric variable. A variable name with no specifier is assumed to be a real numeric variable of single precision. Other signs recognised by most BASICs include: % to specify an integer variable, ! to specify a single precision variable, and # to specify a double precision variable (i.e. the variable can store twice as many significant digits). Here is a fragment of a hypothetical program that uses these signs:

```
70 LET PLAYER$ = "JOHN": REM A STRING
   VARIABLE
80 LET SCORE% = 0: REM AN INTEGER VARIABLE
90 LET PI! = 3.1416: REM A SINGLE PRECISION
   VARIABLE
100 LET AREA# = PI*R*R: REM DOUBLE PRECISION
    VARIABLE
110 LET GOES = 6: REM ASSUMED TO BE SINGLE
    PRECISION REAL
```

Having said that, it must be pointed out that not all BASICs support all these variable types. The Spectrum, for example, does not have integer variables. Integers are simply stored as single precision real numbers. Neither does it support double precision numbers. However, single precision numbers in Spectrum BASIC are calculated to nine significant figures, against only seven significant figures in Microsoft BASIC. The BBC Micro does support variables of the integer type and single precision reals calculated to nine

significant figures. Microsoft BASIC supports double precision variables to 16 significant places.

Computers that do accept integer variables usually allocate two bytes to store the number, which can be in the range −32,768 to 32,767. This range is usually perfectly adequate for such variables as scores, numbers of employees, FOR ... NEXT loop counts and other numbers likely to have only integer values. Since only two bytes are used to store the number, using integer variables if they are available will save on memory, although in many BASICs this is true only for integer arrays, and not for individual variables.

The final part of the Basic Programming course will consider the advantages and disadvantages of BASIC as a language.

**Spectrum Address Book**

This is the full Spectrum version of the Address Book program. Basic Flavours for the Lynx, Dragon 32, BBC Micro, Commodore 64 and Vic-20 will be published in the next issue, and will refer to this listing.

**ZX81 Modifications**

Replace the SAVREC subroutine at line 5600 with:

```
5600 *REM SAVREC*
5610 PRINT "INSERT TAPE,
     PRESS PLAY AND RECORD,
     AND HIT NEWLINE"
5620 INPUT A$
5630 SAVE "ADDBK"
5690 RETURN
```

This will save the whole program together with its data. When using it thereafter, execute it by typing GOTO 10, never by typing RUN, which will set all variables to zero.

Delete the RDINFL subroutine, lines 1400 – 1540, and line 1020.

Each command must have a separate program line, so that line 6770, for example, must be rewritten as:

```
6770 IF A$ = E$ THEN
     PRINT "NEW NAME"
6775 IF A$ = E$ THEN
     INPUT N$(CURR)
```

As printed this program will occupy most of the memory of a 16K ZX81. To save space, REM lines can be deleted and PRINT statements shortened. The number of records for which space is initially reserved can be reduced by modifying the parameter 50 in lines 1110 - - 1160. Line 1170 and all lines referring to XS() — intended for future expansion — may be deleted.

## Basic Flavours

**INKEY$**

On the Lynx in the first program replace lines 110, 120 and 140 by:

```
110 C=KEYN
120 IF (C<>13) AND (C<>32) THEN I=0
140 IF C=13 THEN GOTO 160
```

**CURSOR MOVEMENT**

The third program will RUN, but will not produce the desired result on the Dragon, the BBC, and the Lynx

**PEEK**

On the BBC Micro replace PEEK(S) by ?S

```
1 REM *CREATE DATA FILE*
2 DIM N$(50,30)
3 LET N$(1)="?FIRST"
5 SAVE "NFLD" DATA N$()
6 INPUT"REWIND TAPE, PRESS PLAY, HIT 'ENTER'";A$
7 VERIFY "NFLD" DATA N$()
8 STOP
```

This is the initialising program that creates the array on tape for the first time. When you have run this program, rewind the Data Tape, LOAD the Main Program (listing below) and RUN. You will not need the initialising program again unless you want to create a new address book file.

```
10 REM 'MAINPG'
20 REM *INITIL*
30 GOSUB 1000
40 REM *GREETS*
50 GOSUB 3000
60 FOR M=1 TO 1
70 LET M=0
80 REM *CHOOSE*
90 GOSUB 3500
100 REM *EXECUT*
110 GOSUB 4000
120 IF CHOI=9 THEN LET M=1
130 NEXT M
140 STOP

1000 REM *INITIL* S/R
1010 GOSUB 1100
1020 GOSUB 1400
1030 GOSUB 1600
1090 RETURN

1100 REM *CREARR* S/R
1110 DIM N$(50,30)
1120 DIM M$(50,30)
1130 DIM S$(50,30)
1140 DIM T$(50,15)
1150 DIM C$(50,15)
1160 DIM R$(50,15)
1170 DIM X$(50,30)
1180 DIM B$(30):DIM Z$(30)
1190 DIM U$(30):DIM W$(15)
1210 LET SIZE=0
1220 LET RMOD=0
1230 LET SRTD=1
1240 LET CURR=0
1250 LET Z$="?FIRST"
1260 LET Q$=B$
1300 RETURN

1400 REM *RDINFL* S/R
1405 INPUT"INSERT DATA TAPE, PRESS PLAY, & HIT
     'ENTER'";A$
1410 LOAD "NFLD" DATA N$()
1420 IF N$(1)=Z$ THEN LET Q$=Z$:RETURN
1430 LOAD "MFLD" DATA M$()
1440 LOAD "SFLD" DATA S$()
1450 LOAD "TFLD" DATA T$()
1460 LOAD "CFLD" DATA C$()
1470 LOAD "TELFLD" DATA R$()
1480 LOAD "NDXFLD" DATA X$()
1485 INPUT"STOP THE TAPE, & HIT 'ENTER'";A$
1490 REM 'FLSIZE'
1500 LET SIZE=51
1510 FOR L=1 TO 50
1520 IF N$(L)=B$ THEN LET SIZE=L:LET L=50
1530 NEXT L
1540 RETURN

1600 REM *SETFLG* S/R
1640 IF Q$=Z$ THEN LET SIZE=1
1690 RETURN

3000 REM *GREETS*
3010 CLS
3020 PRINT:PRINT:PRINT:PRINT
3060 PRINT TAB(8);"*WELCOME TO THE*"
3070 PRINT TAB(5);"HOME COMPUTER COURSE*"
3080 PRINT TAB(2);"*COMPUTERISED ADDRESS BOOK*"
3090 PRINT
3100 PRINT TAB(1);"(PRESS SPACE-BAR TO CONTINUE)"
3110 FOR L=1 TO 1
3120 IF INKEY$<>" " THEN LET L=0
3130 NEXT L
3140 CLS
3150 RETURN

3500 REM *CHOOSE* S/R
3520 IF Q$=Z$ THEN GOSUB 3860:RETURN
3540 REM 'CHMENU'
3550 CLS
3560 PRINT"SELECT ONE OF THE FOLLOWING"
3570 PRINT:PRINT:PRINT
3600 PRINT"1. FIND RECORD (FROM NAME)"
3610 PRINT"2. FIND NAMES (FROM INCOMPLETE NAME)"
3620 PRINT"3. FIND RECORDS (FROM TOWN)"
3630 PRINT"4. FIND RECORD (FROM INITIAL)"
3640 PRINT"5. LIST ALL RECORDS"
3650 PRINT"6. ADD NEW RECORD"
3660 PRINT"7. CHANGE RECORD"
3670 PRINT"8. DELETE RECORD"
3680 PRINT"9. EXIT AND SAVE"
3690 PRINT:PRINT
3710 REM 'INCHOI'
3750 PRINT"ENTER CHOICE (1-9)"
3760 FOR L=1 TO 1
3770 FOR I=1 TO 1
3780 LET A$=INKEY$
3790 IF A$="" THEN LET I=0
3800 NEXT I
3810 LET CHOI=CODE A$-48
3820 IF (CHOI<1) OR (CHOI>9) THEN LET L=0
3840 NEXT L
3850 RETURN

3860 REM *FIRSTM* S/R
3870 LET CHOI=6
3880 CLS
3890 PRINT
3900 PRINT TAB(4);"THERE ARE NO RECORDS IN"
3910 PRINT TAB(2);"THE FILE.  YOU WILL HAVE"
3920 PRINT TAB(2);"TO START BY ADDING A RECORD"
3930 PRINT
3940 REM *CONTINUE*
3950 GOSUB 3100
3990 RETURN

4000 REM *EXECUT* S/R
4040 IF CHOI=1 THEN GOSUB 5700
4050 REM 2 IS *FNDNMS*
4060 REM 3 IS *FNDTWN*
4070 REM 4 IS *FNDINT*
4080 REM 5 IS *LSTREC*
4090 IF CHOI=6 THEN GOSUB 4200
4100 IF CHOI=7 THEN GOSUB 6600
4110 IF CHOI=8 THEN GOSUB 7500
4120 IF CHOI=9 THEN GOSUB 5000
4140 RETURN

4200 REM *ADDREC* S/R
4210 CLS
4220 INPUT "ENTER NAME";N$(SIZE)
4230 INPUT "ENTER STREET";S$(SIZE)
4240 INPUT "ENTER TOWN";T$(SIZE)
4250 INPUT "ENTER COUNTY";C$(SIZE)
4260 INPUT "ENTER PHONE NUMBER";R$(SIZE)
4270 LET RMOD=1:LET SRTD=0
4280 LET X$(SIZE)=STR$(SIZE)
4290 LET Q$=""
4300 GOSUB 4500
4310 LET CHOI=0
4320 LET SIZE=SIZE+1
4350 RETURN

4500 REM *MODNAM* S/R
4510 REM CONVERT TO U/CASE
4520 LET D$=N$(SIZE):LET P$=""
4530 FOR L=1 TO LEN(D$)
```

```
4540 LET A$=D$(L)
4550 LET T=CODE A$
4560 IF T>=97 THEN LET T=T-32
4570 LET A$=CHR$ T
4580 LET P$=P$+A$
4590 NEXT L
4600 LET D$=P$:LET P$="":LET A$="":LET T=LEN(D$)
     :LET S=0
4610 REM LOCATE FIRST SPACE
4620 FOR L=1 TO T
4630 IF D$(L)=" " THEN LET S=L:LET L=T
4640 NEXT L
4650 REM REMOVE RUBBISH, PUT FORENAME IN P$
4660 FOR L=1 TO S-1
4670 IF CODE(D$(L)))64 THEN LET P$=P$+D$(L)
4680 NEXT L
4690 REM REMOVE RUBBISH, PUT SURNAME IN A$
4700 FOR L=S+1 TO LEN(D$)
4710 IF CODE(D$(L)))64 THEN LET A$=A$+D$(L)
4720 NEXT L
4730 LET M$(SIZE)=A$+" "+P$
4740 LET P$="":LET A$="":LET S=0
4750 RETURN

5000 REM *EXPROG* S/R
5010 IF (RMOD=0) AND (SRTD=1) THEN RETURN
5020 IF (RMOD=1) AND (SRTD=0) THEN GOSUB 5200
5030 GOSUB 5600
5040 RETURN

5200 REM *SRTREC* S/R
5210 FOR K=1 TO 1
5220 LET S=0
5230 FOR L=1 TO SIZE-2
5240 LET T=L+1
5250 IF M$(L))M$(T) THEN GOSUB 5400
5260 NEXT L
5270 IF S=1 THEN LET K=0
5280 NEXT K
5290 LET SRTD=1
5300 RETURN

5400 REM *SWPREC*
5410 LET U$=N$(L):LET N$(L)=N$(T):LET N$(T)=U$
5420 LET U$=M$(L):LET M$(L)=M$(T):LET M$(T)=U$
5430 LET U$=S$(L):LET S$(L)=S$(T):LET S$(T)=U$
5440 LET W$=T$(L):LET T$(L)=T$(T):LET T$(T)=W$
5450 LET W$=C$(L):LET C$(L)=C$(T):LET C$(T)=W$
5460 LET W$=R$(L):LET R$(L)=R$(T):LET R$(T)=W$
5470 LET X$(L)=STR$(L)
5480 LET X$(T)=STR$(T)
5490 LET S=1
5500 RETURN

5600 REM *SAVREC* S/R
5605 INPUT"INSERT RECORDING TAPE & HiT 'ENTER'";A$
5610 SAVE "NFLD" DATA N$()
5620 SAVE "MFLD" DATA M$()
5630 SAVE "SFLD" DATA S$()
5640 SAVE "TFLD" DATA T$()
5650 SAVE "CFLD" DATA C$()
5660 SAVE "TELFLD" DATA R$()
5670 SAVE "NDXFLD" DATA X$()
5680 INPUT"STOP THE TAPE & HIT 'ENTER'";A$
5690 RETURN

5700 REM *FNDREC* S/R
5710 CLS
5720 IF SRTD=0 THEN GOSUB 5200
5730 PRINT:PRINT
5740 PRINT TAB(5);"SEARCHING FOR A RECORD"
5750 PRINT TAB(12);"BY NAME"
5760 PRINT
5770 PRINT TAB(5);"TYPE IN THE FULL NAME"
5780 PRINT TAB(3);"IN FIRSTNAME SURNAME ORDER"
5790 PRINT:PRINT
5800 INPUT"NAME IS";N$(SIZE)
5810 GOSUB 4500
5820 LET U$=M$(SIZE)
5830 LET BTM=1
5840 LET TP=SIZE-1
5850 FOR X=1 TO 1
5860 LET MD=INT((BTM+TP)/2)
5870 IF M$(MD)<>U$ THEN LET X=0
5880 IF M$(MD)< U$ THEN LET BTM=MD+1
5890 IF M$(MD)> U$ THEN LET TP=MD-1
5900 IF BTM>TP THEN LET X=1
5910 NEXT X
5920 IF BTM>TP THEN LET CURR=0
5930 IF BTM<= TP THEN LET CURR=MD
5940 IF CURR=0 THEN GOSUB 6400:RETURN
5950 CLS
5960 PRINT
5970 PRINT TAB(9);"*RECORD FOUND*"
5980 PRINT
5990 PRINT"NAME:",N$(CURR)
6000 PRINT"STREET:",S$(CURR)
6010 PRINT"TOWN:",T$(CURR)
6020 PRINT"COUNTY:",C$(CURR)
6030 PRINT"PHONE:",R$(CURR)
6040 PRINT
6050 PRINT TAB(3);"PRESS ANY LETTER TO PRINT"
6060 PRINT TAB(3);"OR SPACE-BAR TO CONTINUE"
6070 FOR I=1 TO 1
6080 LET A$=INKEY$
6090 IF A$="" THEN LET I=0
6100 NEXT I
6110 IF A$<>" " THEN GOSUB 6200
6120 RETURN

6200 REM *LSTCUR* S/R
6210 LPRINT
6220 LPRINT"NAME",N$(CURR)
6230 LPRINT"STREET",S$(CURR)
6240 LPRINT"TOWN",T$(CURR)
6250 LPRINT"COUNTY",C$(CURR)
6260 LPRINT"PHONE",R$(CURR)
6270 LPRINT:LPRINT
6280 RETURN

6400 REM *NOTREC* S/R
6410 CLS
6420 PRINT TAB(7);"*RECORD NOT FOUND*"
6430 PRINT TAB(4);"*IN FORM ";N$(SIZE);" *"
6440 PRINT
6450 REM 'CONTINUE'
6460 GOSUB 3100
6470 RETURN

6600 REM *MODREC* S/R
6610 CLS
6620 PRINT:PRINT:PRINT
6630 LET E$=CHR$ 13
6640 PRINT TAB(6);"*TO MODIFY A RECORD*"
6650 PRINT TAB(3);"*FIRST LOCATE THAT RECORD*"
6660 GOSUB 5720
6670 IF CURR=0 THEN RETURN
6680 PRINT
6690 PRINT TAB(10);"MODIFY NAME ?"
6700 PRINT
6710 PRINT TAB(1);"PRESS 'ENTER' TO ENTER NEW NAME"
6720 PRINT TAB(2);"OR SPACE-BAR FOR NEXT FIELD"
6730 FOR I=1 TO 1
6740 LET A$=INKEY$
6750 IF (A$<>E$) AND (A$<>" ") THEN LET I=0
6760 NEXT I
6770 IF A$=E$ THEN INPUT"NEW NAME";N$(CURR)
6780 IF A$=E$ THEN LET RMOD=1
6790 IF A$=E$ THEN LET SRTD=0
6800 IF A$=E$ THEN LET N$(SIZE)=N$(CURR)
6810 IF A$=E$ THEN GOSUB 4500
6820 IF A$=E$ THEN LET M$(CURR)=M$(SIZE)
6830 PRINT
6840 PRINT TAB(8);"MODIFY STREET ?"
6850 PRINT
6860 PRINT TAB(1);"PRESS 'ENTER' TO ENTER NEW STREET"
6870 PRINT TAB(2);"OR SPACE-BAR FOR NEXT FIELD"
6880 FOR I=1 TO 1
6890 LET A$=INKEY$
6900 IF (A$<>E$) AND (A$<>" ") THEN LET I=0
6910 NEXT I
6920 IF A$=E$ THEN LET RMOD=1
6930 IF A$=E$ THEN INPUT"NEW STREET:;S$(CURR)
6940 PRINT
6950 PRINT TAB(10);"MODIFY TOWN ?"
6960 PRINT
6970 PRINT TAB(1);"PRESS 'ENTER' TO ENTER NEW TOWN"
6980 PRINT TAB(2);"OR SPACE-BAR FOR NEXT FIELD"
6990 FOR I=1 TO 1
7010 LET A$=INKEY$
7020 IF (A$<>E$) AND (A$<>" ") THEN LET I=0
7030 NEXT I
7040 IF A$=E$ THEN LET RMOD=1
7050 IF A$=E$ THEN INPUT"NEW TOWN";T$(CURR)
7060 PRINT
7070 PRINT TAB(9);"MODIFY COUNTY ?"
7080 PRINT
7090 PRINT TAB(2);"PRESS 'ENTER' FOR NEW COUNTY"
7100 PRINT TAB(2);"OR SPACE-BAR FOR NEXT FIELD"
7110 FOR I=1 TO 1
7120 LET A$=INKEY$
7130 IF (A$<>E$) AND (A$<>" ") THEN LET I=0
7140 NEXT I
7150 IF A$=E$ THEN LET RMOD=1
7160 IF A$=E$ THEN INPUT"NEW COUNTY";C$(CURR)
7170 PRINT
7180 PRINT TAB(4);"MODIFY PHONE NO. ?"
7190 PRINT
7200 PRINT TAB(1);"PRESS 'ENTER' FOR NEW PHONE NO"
7210 PRINT TAB(2);"OR SPACE-BAR FOR NEXT FIELD"
7220 FOR I=1 TO 1
7230 LET A$=INKEY$
7240 IF (A$<>E$) AND (A$<>" ") THEN LET I=0
7250 NEXT I
7260 IF A$=E$ THEN LET RMOD=1
7270 IF A$=E$ THEN INPUT"NEW NUMBER";R$(CURR)
7280 RETURN

7500 REM *DELREC* S/R
7510 CLS
7520 PRINT:PRINT:PRINT:PRINT
7530 LET E$=CHR$ 13
7540 PRINT TAB(6);"*TO DELETE A RECORD*"
7550 PRINT TAB(3);"*FIRST LOCATE THE DESIRED RECORD*"
7560 GOSUB 5720
7570 IF CURR=0 THEN RETURN
7580 PRINT
7590 PRINT "DO YOU WANT TO DELETE THIS RECORD ?"
7600 PRINT " *WARNING* - NO SECOND CHANCES !"
7610 PRINT
7620 PRINT TAB(5);"PRESS 'ENTER' TO DELETE"
7630 PRINT TAB(4);"OR SPACE-BAR TO CONTINUE"
7640 FOR I=1 TO 1
7650 LET A$=INKEY$
7660 IF (A$<>E$) AND (A$<>" ") THEN LET I=0
7670 NEXT I
7680 IF A$=" " THEN RETURN
7690 FOR L=CURR TO SIZE-2
7700 LET T=L+1
7710 LET N$(L)=N$(T)
7720 LET M$(L)=M$(T)
7730 LET S$(L)=S$(T)
7740 LET T$(L)=T$(T)
7750 LET C$(L)=C$(T)
7760 LET R$(L)=R$(T)
7770 LET X$(L)=X$(T)
7780 NEXT L
7790 LET RMOD=1
7800 LET SIZE=SIZE-1
7810 RETURN
```

TONY LODGE

# University Challenge

## The world's first programmable computer was developed at Manchester University

After the Second World War had ended, Manchester University appointed two new professors. Following his code-breaking work with Colossus, the world's first electromechanical computer, at Bletchley Park, Max Newman became professor of mathematics; and a radar engineer, F C Williams, was appointed head of electrical engineering. Williams brought with him a young assistant, Tom Kilburn, who was familiar with the problems of pulse electronic-memory devices, which he had encountered in his wartime work with radar. Kilburn was later to become the first professor of the new discipline of computer studies at Manchester University.

During a tour of radar establishments in the United States in 1946, Williams had been shown the prototype of the valve computer ENIAC (see page 46), and on his return to England he persuaded the Royal Society to invest £35,000 in a 'Calculating Machine Laboratory' at Manchester. The University was not alone in the race to build a stored program computer. The University of Pennsylvania was constructing the EDVAC, work on the EDSAC was under way at Cambridge University, and the development of the ACE continued at the National Physical Laboratory (see page 88). All these other projects, however, were using a memory store constructed of mercury delay line tubes. The Manchester team were building their machine around a memory device that Williams had invented using a cathode ray tube (CRT). By the autumn of 1947, Williams had succeeded in retaining 2,048 bits for several hours.

Using a 'Williams tube', the Manchester Mark I computer successfully ran a program in June 1948, thus becoming the world's first stored program computer. The Mark I could execute an instruction in 1.2 milliseconds. By using a CRT to store information, the memory had the advantage of being random access, and the contents of the main store or the control register could be visually displayed.

Once the feasibility of using a Williams tube for memory storage was established, an enhanced Mark I was built that could perform work on optics design problems and the generation of prime numbers. The government chief scientist, Sir Ben Lockspeiser, was so impressed by the performance of the computer that he arranged for a commercial version of the Mark I to be built by a local Manchester company. The Ferranti Mark I became available in February 1951, preceding UNIVAC by five months and establishing itself as the first commercially available computer.

An important innovation on the Ferranti Mark I was its ability to modify instructions during processing using another store called the 'B' tube. At the required moment this could add its contents into the control register and thus modify the code of the original instruction. This principle speeded up the processing of programs. IBM used some of the Manchester patents in their early computers, and on a visit to their corporate headquarters in New York, where the company motto (THINK) was emblazoned everywhere, Williams was asked how the Manchester team had succeeded in building a computer where all the resources of IBM had failed. 'We just didn't stop to think too much!' Williams quickly replied.

The arrival of Alan Turing (see page 200) at Manchester in 1948, greatly stimulated programming activities. In 1950 Turing produced the first Manchester programming manual. Two years later, the Manchester team had the idea of building a more compact and economic computer. Their plans were accelerated by the invention of the transistor, and in November 1953 the world's first transistor computer became operational at Manchester.

The late 1950's saw America surging ahead in computer technology, resulting in the British government's decision to invest in a project that would help Britain regain the lead. The Atlas computer, built under the direction of Tom Kilburn, was commissioned in December 1962. It used a 48-bit word with single address format, a 16 Kbyte main store and an eight Kbyte read-only drum memory. Models were sold to the Atomic Energy Research Establishment at Harwell, and British Petroleum, and for many years the Atlas computer was considered to be the most advanced in existence.

**Manchester Mark 1**
Having successfully run a program in June 1948, the Manchester Mark 1 can claim to be the world's first stored program computer. Ferranti, then a local company, were commissioned to develop a commercial version of the computer, which came onto the market in early 1951

COURTESY OF THE UNIVERSITY OF MANCHESTER

# NEXT WEEK

# FREE WITH YOUR LAST ISSUE OF THE HOME COMPUTER COURSE

# ISSUE ONE OF

# THE HOME COMPUTER ADVANCED COURSE

Starting next week, this brand new course takes you on from BASIC to MACHINE CODE programming. In the same clearly-written, easy-to-understand style, The Home Computer Advanced Course will show you how to write software as good as any you can find at your local dealers. And much, much more.

Now is your chance to find out what you and your computer are really capable of.

REMEMBER: Your copy of issue 1 of The Home Computer Advanced Course comes free inside your issue 24 of The Home Computer Course next week.

# Mentathlete

Home computers. Do they send your brain to sleep – or keep your mind on its toes?

At Sinclair, we're in no doubt. To us, a home computer is a mental gym, as important an aid to mental fitness as a set of weights to a body-builder.

Provided, of course, it offers a whole battery of genuine mental challenges.

The Spectrum does just that.

Its education programs turn boring chores into absorbing contests – not learning to spell 'acquiescent', but rescuing a princess from a sorcerer in colour, sound, and movement!

The arcade games would test an all-night arcade freak – they're very fast, very complex, very stimulating.

And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

No other home computer in the world can match the Spectrum challenge – because no other computer has so much software of such outstanding quality to run.

For the Mentathletes of today and tomorrow, the Sinclair Spectrum is gym, apparatus and training schedule, in one neat package. And you can buy one for under £100.

sinclair